

Nº Orden	Apellido y nombre	L.U.	Cantidad de hojas
28	Freund Teodoro	526/15	8

# Organización del Computador 2 Segundo parcial – 29/06/17

## Normas generales

1 (30)	2 (40)	3 (30)	100 (A+)
30	40	30	

- Numere las hojas entregadas. Complete en la primera hoja la cantidad total de hojas entregadas.
- Entregue esta hoja junto al examen, la misma **no** se incluye en la cantidad total de hojas entregadas.
- Está permitido tener los manuales y los apuntes con las listas de instrucciones en el examen. Está prohibido compartir manuales o apuntes entre alumnos durante el examen.
- Cada ejercicio debe realizarse en hojas separadas y numeradas. Debe identificarse cada hoja con nombre, apellido y LU.
- La devolución de los exámenes corregidos es personal. Los pedidos de revisión se realizarán por escrito, antes de retirar el examen corregido del aula.
- Los parciales tienen tres notas: I (Insuficiente): 0 a 59 pts, A- (Aprobado condicional): 60 a 64 pts y A (Aprobado): 65 a 100 pts. No se puede aprobar con A- ambos parciales. Los recuperatorios tienen dos notas: I: 0 a 64 pts y A: 65 a 100 pts.

## Ej. 1. (30 puntos)

Se tiene la siguiente tabla GDT:

Indice	Base	Límite	DB	S	P	L	G	DPL	Tipo
0x32	0x8F8A9000	0x001FF	1	1	1	1	1	3	0xA
0x41	0x00323000	0x00FFF	1	1	1	1	1	3	0x2
0x73	0x03123000	0x0FFFF	1	1	1	1	1	0	0x8
0x92	0x10AB8000	0x00000	1	1	1	1	1	0	0x2

Y el siguiente esquema de paginación:

Rango Lineal	Rango Físico	Atributos
0x00303000 a 0x00C0CFFF	0x3F532000 a 0x3FE3BFFF	read/write, level 0
0x03210000 a 0x10AFFFFF	0x23C16000 a 0x31505FFF	read/write, level 3
0x8F800000 a 0x8FFFFFFF	0xB56A7000 a 0xB5EA6FFF	read/write, level 3

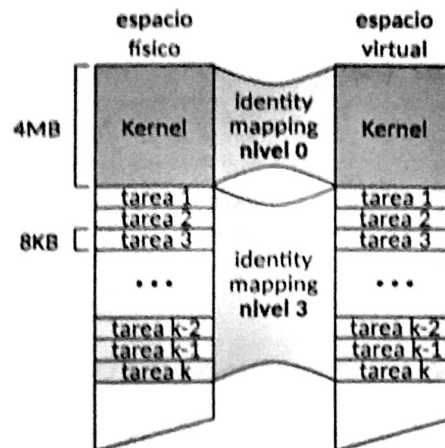
- (12p) a. Especificar todas las entradas de las estructuras necesarias para construir un esquema de paginación. Suponer que todas las entradas no mencionadas son nulas.

- (18p) b. Resolver las siguientes direcciones, de lógica a lineal y a física. Utilizar las estructuras definidas y suponer que cualquier otra estructura no lo está. Si se produjera un error de protección, indicar cuál error y en qué unidad. Definir EPL en todos los casos. El tamaño de todas las operaciones es de 4 bytes.

- I - 0x0190:0x00003221 - CPL 11 - lectura
- II - 0x0208:0x000913AF - CPL 11 - lectura
- III - 0x0398:0x00123831 - CPL 00 - ejecución
- IV - 0x0490:0x00000012 - CPL 00 - escritura
- V - 0x0190:0x00001021 - CPL 00 - ejecución
- VI - 0x0208:0x00833414 - CPL 10 - lectura

## Ej. 2. (40 puntos)

Se tiene un sistema en modo protegido, con paginación activa. El mismo ejecuta K tareas una a una en orden, con un máximo de 500 tareas. Todas las tareas se encuentran alojadas a un área mapeada con *identity mapping* en la cual cualquier otra tarea puede acceder como muestra la siguiente figura:



Cada tarea tiene asignado exactamente 8 KB de espacio direccionable para código, datos y pila. El sistema ejecutará las tareas y detectará cuando una tarea modifica el área de memoria perteneciente a otra tarea, es decir, accede fuera de sus 8 KB. En el caso de detectar una modificación, el sistema ejecutará la función `void fue_modificada(uint32_t tareaModificada, uint32_t tareaModificadora)` indicando el número de la tarea que fue modificada y el número de la tarea que la modificó. Considerar que esta función debe ser ejecutada en algún momento antes de volver a ejecutar la tarea que fue modificada.

- (10p) 1. Indicar los campos relevantes de todas las estructuras involucradas en el sistema para administrar segmentación, paginación, tareas, interrupciones y privilegios. Instanciar las estructuras con datos y explicar su funcionamiento. Describir tanto el esquema de segmentación como el de paginación.
- (20p) 2. Programar en ASM/C la rutina de atención de interrupciones del reloj. Recordar que debe intercambiar las tareas y detectar modificaciones en memoria.
- (10p) 3. ¿Es posible detectar que posición de memoria de una tarea fue modificada por otra? De ser posible, describir en detalle como implementaría este mecanismo o justificar por que no es posible.

Nota: Considerar el uso de los bits **Dirty** y **Accessed** en las *Page Table Entry*

### Ej. 3. (30 puntos)

En un sistema tipo con segmentación y paginación activa, se ejecutan concurrentemente tareas denominadas gusanos. Estos pueden multiplicarse mediante un servicio del sistema. Las tareas llaman al servicio `multiply` y este, crea dos copias de la tarea, una de nombre *derecha* y otra *izquierda*. Para identificar cual es izquierda y cual derecha, el servicio guarda en el registro `al` el byte "I" o "D" respectivamente.

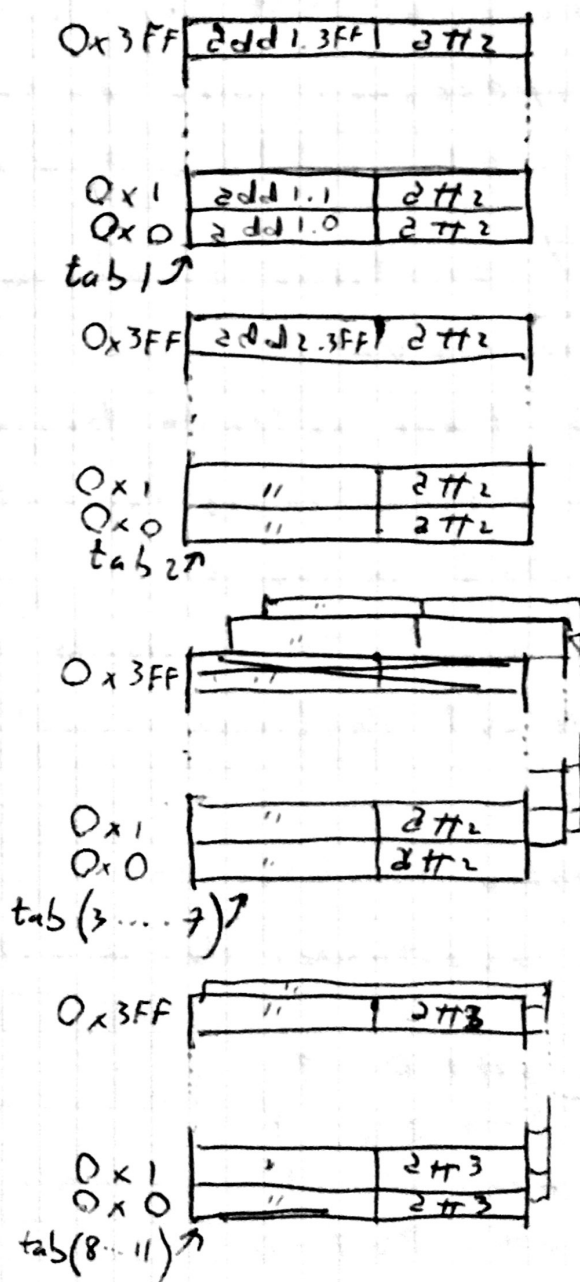
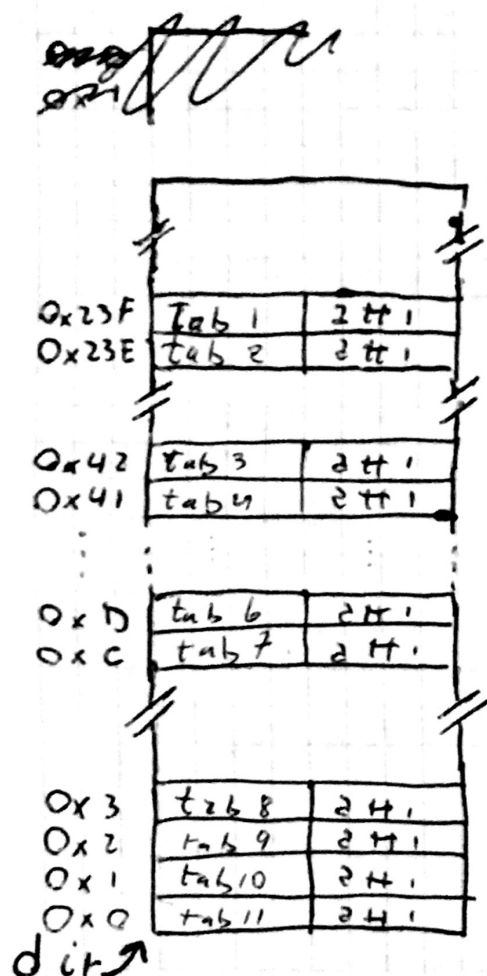
Para duplicar tareas, el servicio cuenta con la función: `tss* duplicar(tss*)`, que toma un puntero a la `tss` de una tarea y retorna una copia de la tarea, pero en otro espacio de memoria físico.

- (5p) 1. Explicar detalladamente que información debe copiar la función `duplicar`.
- (20p) 2. Programar en ASM/C la rutina de atención del servicio `multiply`.
- (5p) 3. ¿Es posible que las tareas duplicadas compartan el mismo CR3? ¿Qué problemas generaría?

Nota: Considerar que la información de la `tss` de la rutina que llama al servicio del sistema contiene la información de la ultima vez que esta fue desalojada.

① Se asume que el Flag L=0 (que no importa), les pongo a los docentes

2. Defina a continuación el directorio de páginas y las tablas necesarias



Cosas a notar:

- dir y tab(01... 11) son las direcciones base del directorio y las tablas, alineadas a 4kB
- entre 4 y 6 ~~Existen~~ Existen tantos números como ~~entre~~ entre 0x41 y 0xD
- en las entradas del directorio, solo están los 20 bits más significativos de la dirección de la tabla
- las addi. ~~es~~ son los 20 bits más significativos para direccionar bien en la tabla i, índice j por ejemplo add8.0 = 0x3F532
- la tab 7 solo tiene seteada desde la entrada 0x210 inclusive
- la tab 3 tiene seteada hasta la 0x2FF, inclusive
- la tab 8 tiene seteadas las entradas ~~de~~ hasta el índice 0xC, inclusive
- la tab 11 tiene seteada desde la 0x303, inclusive
- el resto de las tablas tienen inicializadas todas las entradas
- las ~~atributos~~ atributos son:
  - \* todas las entradas del directorio tienen attr
  - \* con attr.P = 1
    - attr.R/w = 1
    - attr.v/s = 1
    - attr.PWT = 0
    - attr.PCD = 0
    - attr.A = 0
    - attr.ps = 0



Teodoro  
Freund 526/15

2

• Las tablas 1 a 7 tienen 2H.2 con:

2H.2. P = 1

2H.2. P/W = 1

2H.2. U/S = 1

2H.2. PWT = 0

2H.2. PCH = 0

2H.2. A = 0

2H.2. D = 0

2H.2. PAT = 0

2H.2. G = 0

• Las tablas 8 a 11 tienen 2H.3, con todo igual  
a 2H.2, excepto que 2H.3. U/S = 0

b.

(I) 0X0190:0X00003221

indice: 0x32 CPL: 11 RPL: 00 EPL: 11

DPL: 11

Se puede leer (código readable)

Limite OK ✓

0x1FFFFFF > 0x3221

dir lineal = 0x8F8AC221

Está paginado con nivel 3 ✓

dir Física = 0XB5Z53221

✓

② 0x0208:0x000913AF

indice: 0x41 CPL:11 RPL:00 EPL:11 DPL:11

se puede leer (Data, R/W)

límite ok, ~~0x913AF000~~

0xFFFFFFFF > 0x913AF

dir lineal = 0x003B43AF

está paginado con nivel 0, error de privilegios  
en la unidad de paginación (#PF)

③ 0x0398:0x00123831

indice: 0x73 CPL:00 RPL:00 EPL:00 DPL:00

se puede ejecutar (code)

límite ok, 0xFFFFFFFF > 0x00123831

dir lineal: 0x03246831

está paginado con nivel 3 /

dir física: 0x23C4C831

④ 0x0490:0x00000012

indice: 0x92 CPL:00 RPL:00 EPL:00

DPL:00

Se puede escribir (Datos, R/W)

Teodoro  
Frend 526/15

3

Límite Ok,  $0x0000\ 0FFF > 0x0000\ 0012$

dir lineal:  $0x10AB\ 8012$

se encuentra paginada y es escribible

dir Física:  $0x314B\ E012$

⑤  $0x0190: 0x0000\ 1021$

índice  $0x32$  CPL: 00 RPL: 00 EPL: 00 DPL: 11

~~Por~~ el segmento es ejecutable, pero no es  
conforming y  $EPL \neq DPL$

Se produce #GP en la unidad de segmentación

⑥  $0x0208: 0x0083\ 3414$

índice:  $0x41$  CPL: 10 RPL: 00 EPL: 10 DPL: 11

Se puede leer (Datos)

Límite Ok  $0x00FF\ FFFF > 0x0083\ 3414$

dir lineal:  $0x00B56914$

está paginado con modo supervisor, y como

$EPL = 10b = 2$ , se permite el acceso

dir Física:  $0x3FD8\ 5914$

~~el offset~~

el offset en todos los casos es menor que el  
límite por mucho más de 4 Bytes, no hay duda  
con eso

(2) 1.

## Segmentación

Se utilizaría un modelo de segmentación FLAT  
Donde tendríamos una GDT fija con ~~ev~~ tipo  
selecciones de segmentos ~~de~~ de datos y los de  
códigos, uno de cada uno con nivel 3 y <sup>los</sup> otros nivel 0

Todas  
~~Así~~ tendrían los siguientes atributos seteados  
igualmente

Base = 0

Límite = 0x FFFFF (o menos si hay menos memoria)

S = 1

P = 1

AVL = 0

L = 0

DB = 1

G = 1

Los de datos tendrían TIPO = 0x2 (Data, R/W)  
y los de código TIPO = 0x8 (código, no-conforming)

↳ El de código se podría hacer legible, pero  
no tiene sentido, porque es Flat

y DPL sería 0 o 3 dependiendo si es de nivel  
0 o 3, respectivamente

## PAGINACION

Necesitamos paginar los primeros

4MB con identity mapping, es decir necesitamos

una tabla entera con entradas R/W con  $U/S = 0$ ,

el resto de los atributos son como en el ejercicio 1.

Además, necesitamos paginar desde la dirección

$2^{22}$  hasta la  $2^{22} + 8kB \cdot K$  con identity mapping

y  $R/W = 1$ , pero  $U/S = 1$ , es decir nivel 3.

El resto de los atributos son como en el primer ejercicio, ~~excepto~~ Dirty setado en 0.

Dependiendo del diseño, necesitamos 1 o  $K+1$  estructuras de

### TAREAS

esta pinta (todas iguales). Esto no cambia el resto del ejercicio

SE necesitan  $K+1$  entradas de TSS. En la GDT y sus respectivas TSSs, de estas 1 será para la tarea inicial y las otras  $K$  serán para las  $K$  tareas, y deberán ser inicializados con EIP y ESP en posiciones válidas (idealmente en sus páginas) y los selectores de datos y SS en el ~~segundo~~ selector de segmento de datos nivel 3 y CS en el selector de código nivel 3.

Como la paginación es idéntica, se podría compartir un directorio de páginas, pero no es necesario.  
(mismas CR3s)

Además, la pila de nivel 0 (ESP 0) debería caer en los primeros 4MB, y el SS0 debería tener el selector de datos nivel 0.



## Interrupciones

Se debe inicializar la IDT con entradas para las primeras 32 interrupciones, y para la interrupción 32, que llamará a la rutina de atención del reloj. Todos los descriptores deberían ser de interrupciones y tener DPL=0

## Privilegios

Se fue hablando antes, lo importante es que las tareas se ejecuten en nivel 3, y que todo lo que no se puede hacer este en nivel 0

2.

## Rutina de atención de reloj

sched\_tarea\_offset: dd 0x00

sched\_tarea\_selector: dw 0x00 ; Para hacer el salto

global isr32

isr32:

push 2d ; guarda registros

call chequear\_mods ; chequea modificaciones

call sched\_proxima\_tarea ; proxima tarea

cmp 2x, 0

je .nojump ; no haga task switch

mov [sched\_tarea\_selector], 2x

call fin\_intr\_pic1 ; le avisa al PIC

```
    jmp far [sched-tarea-offset] ; hago el taskswitch  
    jmp .end
```

```
nojump:
```

```
    call fin_intr_pic1 ; le aviso al PIC
```

```
    .end:
```

```
    pop ed ; cargo registros
```

```
    iret ; vuelvo
```

### Función para chequear modificaciones

```
void chequear_mods() {
```

```
    void* dir = rcr3(); // guardo CR3 actual
```

```
    dir = ((dir >> 12) << 12); // limpio bits bajos
```

```
    dir = dir + 8; // la primera tabla es del kernel
```

```
    int i = 0;
```

```
    dir = ((dir >> 12) << 12); // las tareas solo usan una tabla
```

```
    for (i = 0; i < 2 * K; i++) { // hay 2 * K PTE válidas
```

```
        int actual = sched_tarea_actual();
```

```
        pag-entry pag-entry* tab = (pag-entry*) dir; // casteo dir
```

```
        // es un arreglo de entradas de paginación
```

```
        if (i/2 != actual and tab[i].D) {
```

```
            fue_modificada(i/2, actual); // aviso si no soy yo  
            // y esta dirty
```

```
        }
```

```
        tab[i].D = 0; // limpio el bit de dirty
```

```
    } tlb_flush(); // flusho la TLB
```

```
    return;
```

```
}
```



Teodoro  
Front 526/15

6

La idea es que antes de cambiar de tarea chequeo todos los bits de DIRTY, si alguno está seteado, y no es el mio, aviso.

Es importante notar que, como hay máximo 500 tareas, y empiezan luego de los primeros 4MB, todas sus entradas entran en la segunda tabla.

Además se asumen las siguientes funciones:

- sched\_proxima\_tarea : retorna el selector de la próxima tarea a correr, o 0 si es la misma. Esto sería  $(actual \% k) + 1$
- fin\_intr\_piol : le avisa al PIC que se retiene la interrupción
- rcr3 : retorna la dirección del directorio de Páginas
- sched\_tarea\_actual : da el Id ( $[1 \dots k]$ ) de la tarea ejecutándose actualmente
- tlb flush : limpia la TLB (guarda y recarga el CR3)

3. Una posible forma de hacerlo sería setear el bit de presente en las entradas de las tablas a 0, entonces, cada vez que se de sea acceder, causaría un PF, y la rutina de atención podría leer qué dirección se intentó leer.

Otra forma, mejor, es setear las páginas como Read-Only, y apagar Write-Protect, entonces, esto genera una excepción solo cuando intentan ~~escribir~~ escribir. Igual que arriba, la rutina leería el código que intentó escribir y guardaría la dirección.

Un problema grave de estas formas es que, la rutina debería setear ~~el~~ el bit de R/W (o P) <sup>antes de retornar</sup> y es decir, que hasta el próximo tick de reloj, la tarea podría acceder a otras direcciones en la misma tabla y el SO. no se enteraría.

Una forma más controlada sería restringir la escritura al uso de un Syscall, pero esto no es lo ideal en ningún sentido.

Teodoro  
Freund 5/6/15

Se asume que la tarea llamo ~~donde~~ vuelve  
como la derecha y la ~~nueva~~  
es la izquierda (fue preguntado)

### ③ 1. Duplicar de bien:

- crear un TSS nuevo, igual al anterior
- agregar una entrada a la GDT para el nuevo TSS
- agregar al scheduler la tarea
- mapear la cantidad de memoria que utiliza la tarea <sup>o copiar</sup> ~~inicial~~ en un directorio de pagina ~~nuevo~~.

Esta memoria se debe mapear con iguales direcciones lineales, pero en un espacio físico nuevo

- Copiar los datos y código, por como se mapea, se copia de una dirección lineal, a la misma lineal del otro directorio de paginación

- Mapeando así, no deben cambiarse los registros (EIP, ESP, ...)

- El posible mapeo de nivel 0 no debería cambiarse, excepto la pila de nivel 0, que debería pedirse e nueva

memoria y remapearse a la misma dir. logica.

- ### 2. La idea sería duplicar, y luego setear la nueva TSS a los valores necesarios para que entre como si no se hubiese pasado, con cuidado de cambiar el posible EAX a otro valor

multiplyate:

```
PUSHAD  
PUSH ESP  
PUSH EAX  
CALL multiplyateC  
ADD ESP, 4  
CALL Fin_intr_pic1  
POPD  
IRET
```

XOR EAX, EAX  
MOV AL, 'D'

```
void multiplicatoc(uint32_tuint32_t esp){
```

```
    tss* actual = ((ret3()) >> 12) << 12; // limpio el CR3
```

```
    tss* nuevo = duplicar(actual); // duplico el TSS
```

```
    // también copia memoria, remap, y creo una  
    // nueva pila de nivel 0
```

```
    // Cambio los valores para que entre justo después  
    // de la llamada al syscall
```

```
    tss->EDI = esp[0];
```

```
    tss->ESI = esp[1];
```

```
    tss->EBP = esp[2]; Para cambiar el ESP a esp[2]
```

```
    tss->ESP = esp[11]; // esta es la pila de nivel 3
```

```
    tss->EBX = esp[4];
```

```
    tss->EDX = esp[5];
```

```
    tss->ECX = esp[6];
```

```
    tss->EAX = 'I';
```

```
    tss->EIP = esp[8];
```

```
    // el CR3 está setando Bien, SS2, SS1, ESP2, ESP1 no
```

```
    // importan (los seteo duplicar) SS0 y ESP0 los seteo
```

```
    // duplicar, LDT no importa
```

```
tss->LDT
```

```
    tss->GS = Datos Usuario; // se asume que el sistema
```

```
    tss->FS = Datos Usuario; // tipo tiene segmentación
```

```
    tss->DS = Datos Usuario; // Flat
```

```
    tss->SS = esp[12];
```

```
    tss->CS = esp[9];
```

```
    tss->ES = Datos Usuario;
```

```
    tss->EFlags = esp[10];
```

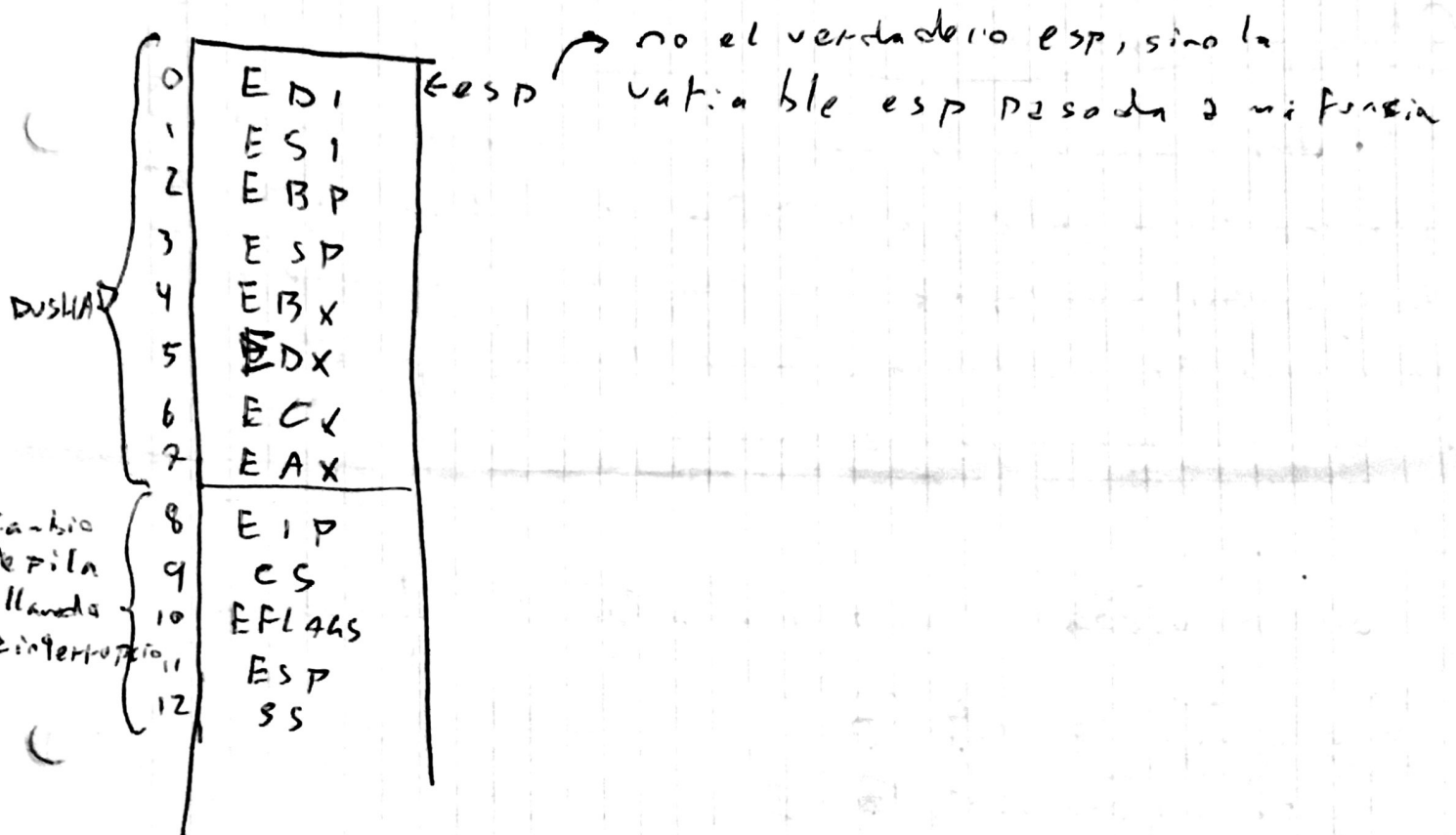
```
    return;
```

```
}
```



Por como funciona multiplicate C, cuando se  
paga un jmp far a la tarea nueva, continuara  
exactamente luego del llamado al syscall

Para entender mejor, se dibuja la pila  
debajo del esp que se le pasa a multiplicate C



3. Las tareas podrian compartir el mismo CR3, pero  
traeria los siguientes problemas, entre otros:

- \* no habria proteccion (aislamiento) entre tareas
- \* a la hora de duplicar, no podriamos mapear  
las distintas direcciones fisicas a iguales  
lineales, y deberiamos haber cambiado todas  
las registras necesarias, principalmente EIP, ESP.

Pero, además, si la tarea usase algún otro registro ~~para acceder~~ para acceder a memoria, habría que recalcular ~~la~~ cual era la posición intencionada, o en su defecto, implementar algún tipo de sistema Copy-on-Write y recalcular cuando se desee escribir.

### Otra posible solución para el 3.2

Podría haber sido hacer la entrada al syscall por una task gate, con DPL = 3, P = 1 y TIPO = 5. Luego haría un task switch con otro TSS que, inicialmente saltaría a [multiply], y esta función se haría como 2 continuación.

multiply

```
call prev-TSStask ; el puntero al al TSS que me  
push eax ; invoco  
mov [eax+OFF_EAX], 'D'  
call duplicar  
mov [eax+OFF_EAX], 'I'  
iret
```

JMP [multiply]

No utilice esta solución porque no está segura de un par de cosas (y es algo que no sé), pero si las cosas funcionan como creo que funcionan, el iret debería volver a la tarea que lo invoca, con el contexto igual (EAX = 'D') y, cuando el scheduler haga un task switch a la tarea creada, vuelve idéntico a la otra (con EAX = 'I')